Memory Management Report

Tingyu Shi(400253854)
Jiacheng Wu(400207981)

September 14, 2022

1 Analysis

1.1 Three algorithms without Compaction

For the sample output, please check section 2.1.
For the code, please check section 3.1.
he following are results of 10 trials.

	First Fit	Best Fit	Worst Fit
Trial 1	72	72	72
Trial 2	132	132	132
Trial 3	120	120	120
Trial 4	232	232	232
Trial 5	184	184	184
Trial 6	176	176	176
Trial 7	160	160	160
Trial 8	96	96	96
Trial 9	48	48	48
Trial 10	148	148	148

Table 1: Hole Size(KB) Information after removing 10%

Average Hole size after removing 10% is 136.8(KB)

	First Fit	Best Fit	Worst Fit
Trial 1	8	8	8
Trial 2	8	20	56
Trial 3	20	20	20
Trial 4	24	24	24
Trial 5	24	0	44
Trial 6	16	16	40
Trial 7	36	36	36
Trial 8	20	20	60
Trial 9	28	28	28
Trial 10	60	12	60

Table 2: Hole Size(KB) Information after refilling

Average Hole size after refilling(First Fit) is 24.4(KB)
Average Hole size after refilling(Best Fit) is 18.4(KB)
Average Hole size after refilling(Worst Fit) is 37.6(KB)

Frist Fit memory utilization rate

$$= \frac{136.8 - 24.4}{136.8} = 82.16\%$$

Best Fit memory utilization rate

$$= \frac{136.8 - 18.4}{136.8} = 86.55\%$$

Worst Fit memory utilization rate

$$= \frac{136.8 - 37.6}{136.8} = 72.51\%$$

Conclusion: When there is no compaction, $Best\ Fit > First\ Fit > Worst\ Fit$. The experiment result may be caused by the following factors:

- Best Fit tries to find the smallest hole which can fit the process. As a result, the hold size left is the smallest.
- The experiment is designed in a way that each process's size is a multiple of 4KB and the range is [4KB, 100KB]. This may lead to inaccurate experiment result. If we can use random process size, the result may be more accurate.

1.2 Three algorithms with Compaction

For the sample output, please check section 2.2.
For the code, please check section 3.2.
The following are results of 10 trials.

	First Fit	Best Fit	Worst Fit
Trial 1	56	56	56
Trial 2	120	120	120
Trial 3	168	168	168
Trial 4	148	148	148
Trial 5	100	100	100
Trial 6	84	84	84
Trial 7	184	184	184
Trial 8	140	140	140
Trial 9	144	144	144
Trial 10	92	92	92

Table 3: Hole Size(KB) Information after removing 10% and after compaction

	First Fit	Best Fit	Worst Fit
Trial 1	0	0	0
Trial 2	20	20	20
Trial 3	12	12	12
Trial 4	24	24	24
Trial 5	4	4	4
Trial 6	0	0	0
Trial 7	8	8	8
Trial 8	8	8	8
Trial 9	12	12	12
Trial 10	12	12	12

Table 4: Hole Size(KB) Information after refilling

For each trial, after removing 10% and compaction, there is only one hole in the memory and the hole sizes are the same because the program removes the same 10% of the processes. Since there is only one hole in the memory, three algorithms perform in the same way. As as a result, in table 4, three algorithms show the same hole sizes for each trial after refilling.
Conclusion: Three algorithms perform the same after compaction as there is only one hole in the memory after compaction.

2 Sample Outputs

2.1 Sample output of three algorithms without Compaction

```
======================Memory Information after removing 10 percent======================

================Memory Information of First Fit algorithm================
size: 1MB
Memory starting address: 0x160008000
Memory ending address:   0x160107fff

++++++++++Process Information++++++++++
----------------------------------------------
Process0 starting address: 0x160008000
Process0 size: 56 KB
----------------------------------------------
Process1 starting address: 0x160016000
Process1 size: 16 KB
----------------------------------------------
Process2 starting address: 0x16001a000
Process2 size: 56 KB
----------------------------------------------
Process3 starting address: 0x160028000
Process3 size: 44 KB
----------------------------------------------
Process4 starting address: 0x160033000
Process4 size: 20 KB
----------------------------------------------
Process5 starting address: 0x160038000
Process5 size: 32 KB
----------------------------------------------
Process6 starting address: 0x160040000
Process6 size: 20 KB
----------------------------------------------
Process7 starting address: 0x160045000
Process7 size: 16 KB
----------------------------------------------
Process9 starting address: 0x16005d000
Process9 size: 32 KB
----------------------------------------------
Process10 starting address: 0x160065000
Process10 size: 60 KB
----------------------------------------------
Process11 starting address: 0x160074000
Process11 size: 64 KB
----------------------------------------------
Process12 starting address: 0x160084000
Process12 size: 12 KB
----------------------------------------------
Process14 starting address: 0x160093000
```

```
Process14 size: 100 KB
------------------------------------------------
Process15 starting address: 0x1600ac000
Process15 size: 56 KB
------------------------------------------------
Process16 starting address: 0x1600ba000
Process16 size: 92 KB
------------------------------------------------
Process17 starting address: 0x1600d1000
Process17 size: 52 KB
------------------------------------------------
Process18 starting address: 0x1600de000
Process18 size: 44 KB
------------------------------------------------
Process19 starting address: 0x1600e9000
Process19 size: 40 KB
------------------------------------------------

+++++++++++Hole Information+++++++++++
--------------------------------------------
Hole0 starting address: 0x160049000
Hole0 size: 80 KB
--------------------------------------------
Hole1 starting address: 0x160087000
Hole1 size: 48 KB
--------------------------------------------
Hole2 starting address: 0x1600f3000
Hole2 size: 84 KB
--------------------------------------------

================Memory Information of Best Fit algorithm================
size: 1MB
Memory starting address: 0x160108000
Memory ending address:   0x160207fff

++++++++++Process Information++++++++++
------------------------------------------------
Process0 starting address: 0x160108000
Process0 size: 56 KB
------------------------------------------------
Process1 starting address: 0x160116000
Process1 size: 16 KB
------------------------------------------------
Process2 starting address: 0x16011a000
Process2 size: 56 KB
------------------------------------------------
Process3 starting address: 0x160128000
Process3 size: 44 KB
------------------------------------------------
Process4 starting address: 0x160133000
```

```
Process4 size: 20 KB
----------------------------------------
Process5 starting address: 0x160138000
Process5 size: 32 KB
----------------------------------------
Process6 starting address: 0x160140000
Process6 size: 20 KB
----------------------------------------
Process7 starting address: 0x160145000
Process7 size: 16 KB
----------------------------------------
Process9 starting address: 0x16015d000
Process9 size: 32 KB
----------------------------------------
Process10 starting address: 0x160165000
Process10 size: 60 KB
----------------------------------------
Process11 starting address: 0x160174000
Process11 size: 64 KB
----------------------------------------
Process12 starting address: 0x160184000
Process12 size: 12 KB
----------------------------------------
Process14 starting address: 0x160193000
Process14 size: 100 KB
----------------------------------------
Process15 starting address: 0x1601ac000
Process15 size: 56 KB
----------------------------------------
Process16 starting address: 0x1601ba000
Process16 size: 92 KB
----------------------------------------
Process17 starting address: 0x1601d1000
Process17 size: 52 KB
----------------------------------------
Process18 starting address: 0x1601de000
Process18 size: 44 KB
----------------------------------------
Process19 starting address: 0x1601e9000
Process19 size: 40 KB
----------------------------------------

+++++++++++Hole Information+++++++++++
------------------------------------
Hole0 starting address: 0x160149000
Hole0 size: 80 KB
------------------------------------
Hole1 starting address: 0x160187000
Hole1 size: 48 KB
------------------------------------
```

```
Hole2 starting address: 0x1601f3000
Hole2 size: 84 KB
-------------------------------------------

================Memory Information of Worst Fit algorithm================
size: 1MB
Memory starting address: 0x160208000
Memory ending address:   0x160307fff

++++++++++Process Information++++++++++
-----------------------------------------------
Process0 starting address: 0x160208000
Process0 size: 56 KB
-----------------------------------------------
Process1 starting address: 0x160216000
Process1 size: 16 KB
-----------------------------------------------
Process2 starting address: 0x16021a000
Process2 size: 56 KB
-----------------------------------------------
Process3 starting address: 0x160228000
Process3 size: 44 KB
-----------------------------------------------
Process4 starting address: 0x160233000
Process4 size: 20 KB
-----------------------------------------------
Process5 starting address: 0x160238000
Process5 size: 32 KB
-----------------------------------------------
Process6 starting address: 0x160240000
Process6 size: 20 KB
-----------------------------------------------
Process7 starting address: 0x160245000
Process7 size: 16 KB
-----------------------------------------------
Process9 starting address: 0x16025d000
Process9 size: 32 KB
-----------------------------------------------
Process10 starting address: 0x160265000
Process10 size: 60 KB
-----------------------------------------------
Process11 starting address: 0x160274000
Process11 size: 64 KB
-----------------------------------------------
Process12 starting address: 0x160284000
Process12 size: 12 KB
-----------------------------------------------
Process14 starting address: 0x160293000
Process14 size: 100 KB
-----------------------------------------------
```

```
Process15 starting address: 0x1602ac000
Process15 size: 56 KB
--------------------------------------------
Process16 starting address: 0x1602ba000
Process16 size: 92 KB
--------------------------------------------
Process17 starting address: 0x1602d1000
Process17 size: 52 KB
--------------------------------------------
Process18 starting address: 0x1602de000
Process18 size: 44 KB
--------------------------------------------
Process19 starting address: 0x1602e9000
Process19 size: 40 KB
--------------------------------------------

++++++++++Hole Information++++++++++
----------------------------------------
Hole0 starting address: 0x160249000
Hole0 size: 80 KB
----------------------------------------
Hole1 starting address: 0x160287000
Hole1 size: 48 KB
----------------------------------------
Hole2 starting address: 0x1602f3000
Hole2 size: 84 KB
----------------------------------------

==========================Memory Information after refilling==========================

==============Memory Information of First Fit algorithm===============
size: 1MB
Memory starting address: 0x160008000
Memory ending address:   0x160107fff

++++++++++Process Information++++++++++
--------------------------------------------
Process0 starting address: 0x160008000
Process0 size: 56 KB
--------------------------------------------
Process1 starting address: 0x160016000
Process1 size: 16 KB
--------------------------------------------
Process2 starting address: 0x16001a000
Process2 size: 56 KB
--------------------------------------------
Process3 starting address: 0x160028000
Process3 size: 44 KB
--------------------------------------------
Process4 starting address: 0x160033000
```

```
Process4 size: 20 KB
--------------------------------------------------
Process5 starting address: 0x160038000
Process5 size: 32 KB
--------------------------------------------------
Process6 starting address: 0x160040000
Process6 size: 20 KB
--------------------------------------------------
Process7 starting address: 0x160045000
Process7 size: 16 KB
--------------------------------------------------
Process9 starting address: 0x16005d000
Process9 size: 32 KB
--------------------------------------------------
Process10 starting address: 0x160065000
Process10 size: 60 KB
--------------------------------------------------
Process11 starting address: 0x160074000
Process11 size: 64 KB
--------------------------------------------------
Process12 starting address: 0x160084000
Process12 size: 12 KB
--------------------------------------------------
Process14 starting address: 0x160093000
Process14 size: 100 KB
--------------------------------------------------
Process15 starting address: 0x1600ac000
Process15 size: 56 KB
--------------------------------------------------
Process16 starting address: 0x1600ba000
Process16 size: 92 KB
--------------------------------------------------
Process17 starting address: 0x1600d1000
Process17 size: 52 KB
--------------------------------------------------
Process18 starting address: 0x1600de000
Process18 size: 44 KB
--------------------------------------------------
Process19 starting address: 0x1600e9000
Process19 size: 40 KB
--------------------------------------------------
Process21 starting address: 0x160049000
Process21 size: 56 KB
--------------------------------------------------
Process22 starting address: 0x160057000
Process22 size: 16 KB
--------------------------------------------------
Process23 starting address: 0x1600f3000
Process23 size: 56 KB
--------------------------------------------------
```

```
Process24 starting address: 0x160087000
Process24 size: 44 KB
-------------------------------------------
Process25 starting address: 0x160101000
Process25 size: 20 KB
-------------------------------------------

++++++++++Hole Information++++++++++
---------------------------------------
Hole0 starting address: 0x16005b000
Hole0 size: 8 KB
---------------------------------------
Hole1 starting address: 0x160092000
Hole1 size: 4 KB
---------------------------------------
Hole2 starting address: 0x160106000
Hole2 size: 8 KB
---------------------------------------

================Memory Information of Best Fit algorithm================
size: 1MB
Memory starting address: 0x160108000
Memory ending address:   0x160207fff

++++++++++Process Information++++++++++
-------------------------------------------
Process0 starting address: 0x160108000
Process0 size: 56 KB
-------------------------------------------
Process1 starting address: 0x160116000
Process1 size: 16 KB
-------------------------------------------
Process2 starting address: 0x16011a000
Process2 size: 56 KB
-------------------------------------------
Process3 starting address: 0x160128000
Process3 size: 44 KB
-------------------------------------------
Process4 starting address: 0x160133000
Process4 size: 20 KB
-------------------------------------------
Process5 starting address: 0x160138000
Process5 size: 32 KB
-------------------------------------------
Process6 starting address: 0x160140000
Process6 size: 20 KB
-------------------------------------------
Process7 starting address: 0x160145000
Process7 size: 16 KB
-------------------------------------------
```

```
Process9 starting address: 0x16015d000
Process9 size: 32 KB
```

```
Process10 starting address: 0x160165000
Process10 size: 60 KB
```

```
Process11 starting address: 0x160174000
Process11 size: 64 KB
```

```
Process12 starting address: 0x160184000
Process12 size: 12 KB
```

```
Process14 starting address: 0x160193000
Process14 size: 100 KB
```

```
Process15 starting address: 0x1601ac000
Process15 size: 56 KB
```

```
Process16 starting address: 0x1601ba000
Process16 size: 92 KB
```

```
Process17 starting address: 0x1601d1000
Process17 size: 52 KB
```

```
Process18 starting address: 0x1601de000
Process18 size: 44 KB
```

```
Process19 starting address: 0x1601e9000
Process19 size: 40 KB
```

```
Process21 starting address: 0x160149000
Process21 size: 56 KB
```

```
Process22 starting address: 0x160157000
Process22 size: 16 KB
```

```
Process23 starting address: 0x1601f3000
Process23 size: 56 KB
```

```
Process24 starting address: 0x160187000
Process24 size: 44 KB
```

```
Process25 starting address: 0x160201000
Process25 size: 20 KB
```

```
+++++++++++Hole Information+++++++++++
```

```
Hole0 starting address: 0x16015b000
Hole0 size: 8 KB
```

```
-----------------------------------------
Hole1 starting address: 0x160192000
Hole1 size: 4 KB
-----------------------------------------
Hole2 starting address: 0x160206000
Hole2 size: 8 KB
-----------------------------------------

===============Memory Information of Worst Fit algorithm===============
size: 1MB
Memory starting address: 0x160208000
Memory ending address:   0x160307fff

++++++++++Process Information++++++++++
---------------------------------------------
Process0 starting address: 0x160208000
Process0 size: 56 KB
---------------------------------------------
Process1 starting address: 0x160216000
Process1 size: 16 KB
---------------------------------------------
Process2 starting address: 0x16021a000
Process2 size: 56 KB
---------------------------------------------
Process3 starting address: 0x160228000
Process3 size: 44 KB
---------------------------------------------
Process4 starting address: 0x160233000
Process4 size: 20 KB
---------------------------------------------
Process5 starting address: 0x160238000
Process5 size: 32 KB
---------------------------------------------
Process6 starting address: 0x160240000
Process6 size: 20 KB
---------------------------------------------
Process7 starting address: 0x160245000
Process7 size: 16 KB
---------------------------------------------
Process9 starting address: 0x16025d000
Process9 size: 32 KB
---------------------------------------------
Process10 starting address: 0x160265000
Process10 size: 60 KB
---------------------------------------------
Process11 starting address: 0x160274000
Process11 size: 64 KB
---------------------------------------------
Process12 starting address: 0x160284000
Process12 size: 12 KB
```

```
-----------------------------------------------
Process14 starting address: 0x160293000
Process14 size: 100 KB
-----------------------------------------------
Process15 starting address: 0x1602ac000
Process15 size: 56 KB
-----------------------------------------------
Process16 starting address: 0x1602ba000
Process16 size: 92 KB
-----------------------------------------------
Process17 starting address: 0x1602d1000
Process17 size: 52 KB
-----------------------------------------------
Process18 starting address: 0x1602de000
Process18 size: 44 KB
-----------------------------------------------
Process19 starting address: 0x1602e9000
Process19 size: 40 KB
-----------------------------------------------
Process21 starting address: 0x1602f3000
Process21 size: 56 KB
-----------------------------------------------
Process22 starting address: 0x160249000
Process22 size: 16 KB
-----------------------------------------------
Process23 starting address: 0x16024d000
Process23 size: 56 KB
-----------------------------------------------
Process24 starting address: 0x160287000
Process24 size: 44 KB
-----------------------------------------------
Process25 starting address: 0x160301000
Process25 size: 20 KB
-----------------------------------------------

+++++++++++Hole Information++++++++++++
-------------------------------------------
Hole0 starting address: 0x16025b000
Hole0 size: 8 KB
-------------------------------------------
Hole1 starting address: 0x160292000
Hole1 size: 4 KB
-------------------------------------------
Hole2 starting address: 0x160306000
Hole2 size: 8 KB
-------------------------------------------
```

2.2 Sample output of three algorithms with Compaction

```
====================Memory Information after removing 10 percent====================

===============Memory Information of First Fit algorithm===============
size: 1MB
Memory starting address: 0x150008000
Memory ending address:   0x150107fff

++++++++++Process Information++++++++++
------------------------------------------
Process0 starting address: 0x150008000
Process0 size: 16 KB
------------------------------------------
Process2 starting address: 0x150019000
Process2 size: 4 KB
------------------------------------------
Process3 starting address: 0x15001a000
Process3 size: 72 KB
------------------------------------------
Process4 starting address: 0x15002c000
Process4 size: 72 KB
------------------------------------------
Process5 starting address: 0x15003e000
Process5 size: 48 KB
------------------------------------------
Process6 starting address: 0x15004a000
Process6 size: 4 KB
------------------------------------------
Process7 starting address: 0x15004b000
Process7 size: 12 KB
------------------------------------------
Process8 starting address: 0x15004e000
Process8 size: 76 KB
------------------------------------------
Process9 starting address: 0x150061000
Process9 size: 40 KB
------------------------------------------
Process10 starting address: 0x15006b000
Process10 size: 8 KB
------------------------------------------
Process11 starting address: 0x15006d000
Process11 size: 28 KB
------------------------------------------
Process13 starting address: 0x15007c000
Process13 size: 84 KB
------------------------------------------
Process14 starting address: 0x150091000
Process14 size: 24 KB
------------------------------------------
```

```
Process15 starting address: 0x150097000
Process15 size: 48 KB
---------------------------------------------
Process16 starting address: 0x1500a3000
Process16 size: 8 KB
---------------------------------------------
Process17 starting address: 0x1500a5000
Process17 size: 72 KB
---------------------------------------------
Process18 starting address: 0x1500b7000
Process18 size: 92 KB
---------------------------------------------
Process19 starting address: 0x1500ce000
Process19 size: 64 KB
---------------------------------------------
Process20 starting address: 0x1500de000
Process20 size: 60 KB
---------------------------------------------
Process21 starting address: 0x1500ed000
Process21 size: 76 KB
---------------------------------------------
Process22 starting address: 0x150100000
Process22 size: 32 KB
---------------------------------------------

++++++++++Hole Information++++++++++
-----------------------------------------
Hole0 starting address: 0x15000c000
Hole0 size: 52 KB
-----------------------------------------
Hole1 starting address: 0x150074000
Hole1 size: 32 KB
-----------------------------------------

================Memory Information of Best Fit algorithm================
size: 1MB
Memory starting address: 0x150108000
Memory ending address:   0x150207fff

++++++++++Process Information++++++++++
---------------------------------------------
Process0 starting address: 0x150108000
Process0 size: 16 KB
---------------------------------------------
Process2 starting address: 0x150119000
Process2 size: 4 KB
---------------------------------------------
Process3 starting address: 0x15011a000
Process3 size: 72 KB
---------------------------------------------
```

```
Process4 starting address: 0x15012c000
Process4 size: 72 KB
```

```
Process5 starting address: 0x15013e000
Process5 size: 48 KB
```

```
Process6 starting address: 0x15014a000
Process6 size: 4 KB
```

```
Process7 starting address: 0x15014b000
Process7 size: 12 KB
```

```
Process8 starting address: 0x15014e000
Process8 size: 76 KB
```

```
Process9 starting address: 0x150161000
Process9 size: 40 KB
```

```
Process10 starting address: 0x15016b000
Process10 size: 8 KB
```

```
Process11 starting address: 0x15016d000
Process11 size: 28 KB
```

```
Process13 starting address: 0x15017c000
Process13 size: 84 KB
```

```
Process14 starting address: 0x150191000
Process14 size: 24 KB
```

```
Process15 starting address: 0x150197000
Process15 size: 48 KB
```

```
Process16 starting address: 0x1501a3000
Process16 size: 8 KB
```

```
Process17 starting address: 0x1501a5000
Process17 size: 72 KB
```

```
Process18 starting address: 0x1501b7000
Process18 size: 92 KB
```

```
Process19 starting address: 0x1501ce000
Process19 size: 64 KB
```

```
Process20 starting address: 0x1501de000
Process20 size: 60 KB
```

```
Process21 starting address: 0x1501ed000
Process21 size: 76 KB
```

```
---------------------------------------------
Process22 starting address: 0x150200000
Process22 size: 32 KB
---------------------------------------------

+++++++++++Hole Information+++++++++++
-----------------------------------------
Hole0 starting address: 0x15010c000
Hole0 size: 52 KB
-----------------------------------------
Hole1 starting address: 0x150174000
Hole1 size: 32 KB
-----------------------------------------

================Memory Information of Worst Fit algorithm================
size: 1MB
Memory starting address: 0x150208000
Memory ending address:   0x150307fff

++++++++++Process Information++++++++++
---------------------------------------------
Process0 starting address: 0x150208000
Process0 size: 16 KB
---------------------------------------------
Process2 starting address: 0x150219000
Process2 size: 4 KB
---------------------------------------------
Process3 starting address: 0x15021a000
Process3 size: 72 KB
---------------------------------------------
Process4 starting address: 0x15022c000
Process4 size: 72 KB
---------------------------------------------
Process5 starting address: 0x15023e000
Process5 size: 48 KB
---------------------------------------------
Process6 starting address: 0x15024a000
Process6 size: 4 KB
---------------------------------------------
Process7 starting address: 0x15024b000
Process7 size: 12 KB
---------------------------------------------
Process8 starting address: 0x15024e000
Process8 size: 76 KB
---------------------------------------------
Process9 starting address: 0x150261000
Process9 size: 40 KB
---------------------------------------------
Process10 starting address: 0x15026b000
Process10 size: 8 KB
```

```
------------------------------------------
Process11 starting address: 0x15026d000
Process11 size: 28 KB
------------------------------------------
Process13 starting address: 0x15027c000
Process13 size: 84 KB
------------------------------------------
Process14 starting address: 0x150291000
Process14 size: 24 KB
------------------------------------------
Process15 starting address: 0x150297000
Process15 size: 48 KB
------------------------------------------
Process16 starting address: 0x1502a3000
Process16 size: 8 KB
------------------------------------------
Process17 starting address: 0x1502a5000
Process17 size: 72 KB
------------------------------------------
Process18 starting address: 0x1502b7000
Process18 size: 92 KB
------------------------------------------
Process19 starting address: 0x1502ce000
Process19 size: 64 KB
------------------------------------------
Process20 starting address: 0x1502de000
Process20 size: 60 KB
------------------------------------------
Process21 starting address: 0x1502ed000
Process21 size: 76 KB
------------------------------------------
Process22 starting address: 0x150300000
Process22 size: 32 KB
------------------------------------------

++++++++++++Hole Information++++++++++++
--------------------------------------
Hole0 starting address: 0x15020c000
Hole0 size: 52 KB
--------------------------------------
Hole1 starting address: 0x150274000
Hole1 size: 32 KB
--------------------------------------


==================Memory Information after Compaction==========================

================Memory Information of First Fit algorithm================
size: 1MB
Memory starting address: 0x150008000
```

```
Memory ending address:   0x150107fff

++++++++++Process Information++++++++++
----------------------------------------
Process0 starting address: 0x150008000
Process0 size: 16 KB
----------------------------------------
Process2 starting address: 0x15000c000
Process2 size: 4 KB
----------------------------------------
Process3 starting address: 0x15000d000
Process3 size: 72 KB
----------------------------------------
Process4 starting address: 0x15001f000
Process4 size: 72 KB
----------------------------------------
Process5 starting address: 0x150031000
Process5 size: 48 KB
----------------------------------------
Process6 starting address: 0x15003d000
Process6 size: 4 KB
----------------------------------------
Process7 starting address: 0x15003e000
Process7 size: 12 KB
----------------------------------------
Process8 starting address: 0x150041000
Process8 size: 76 KB
----------------------------------------
Process9 starting address: 0x150054000
Process9 size: 40 KB
----------------------------------------
Process10 starting address: 0x15005e000
Process10 size: 8 KB
----------------------------------------
Process11 starting address: 0x150060000
Process11 size: 28 KB
----------------------------------------
Process13 starting address: 0x150067000
Process13 size: 84 KB
----------------------------------------
Process14 starting address: 0x15007c000
Process14 size: 24 KB
----------------------------------------
Process15 starting address: 0x150082000
Process15 size: 48 KB
----------------------------------------
Process16 starting address: 0x15008e000
Process16 size: 8 KB
----------------------------------------
Process17 starting address: 0x150090000
```

```
Process17 size: 72 KB
-------------------------------------------------
Process18 starting address: 0x1500a2000
Process18 size: 92 KB
-------------------------------------------------
Process19 starting address: 0x1500b9000
Process19 size: 64 KB
-------------------------------------------------
Process20 starting address: 0x1500c9000
Process20 size: 60 KB
-------------------------------------------------
Process21 starting address: 0x1500d8000
Process21 size: 76 KB
-------------------------------------------------
Process22 starting address: 0x1500eb000
Process22 size: 32 KB
-------------------------------------------------

+++++++++++Hole Information+++++++++++
-------------------------------------------------
Hole0 starting address: 0x1500f3000
Hole0 size: 84 KB
-------------------------------------------------

================Memory Information of Best Fit algorithm================
size: 1MB
Memory starting address: 0x150108000
Memory ending address:   0x150207fff

++++++++++Process Information++++++++++
-------------------------------------------------
Process0 starting address: 0x150108000
Process0 size: 16 KB
-------------------------------------------------
Process2 starting address: 0x15010c000
Process2 size: 4 KB
-------------------------------------------------
Process3 starting address: 0x15010d000
Process3 size: 72 KB
-------------------------------------------------
Process4 starting address: 0x15011f000
Process4 size: 72 KB
-------------------------------------------------
Process5 starting address: 0x150131000
Process5 size: 48 KB
-------------------------------------------------
Process6 starting address: 0x15013d000
Process6 size: 4 KB
-------------------------------------------------
Process7 starting address: 0x15013e000
```

```
Process7 size: 12 KB
------------------------------------------
Process8 starting address: 0x150141000
Process8 size: 76 KB
------------------------------------------
Process9 starting address: 0x150154000
Process9 size: 40 KB
------------------------------------------
Process10 starting address: 0x15015e000
Process10 size: 8 KB
------------------------------------------
Process11 starting address: 0x150160000
Process11 size: 28 KB
------------------------------------------
Process13 starting address: 0x150167000
Process13 size: 84 KB
------------------------------------------
Process14 starting address: 0x15017c000
Process14 size: 24 KB
------------------------------------------
Process15 starting address: 0x150182000
Process15 size: 48 KB
------------------------------------------
Process16 starting address: 0x15018e000
Process16 size: 8 KB
------------------------------------------
Process17 starting address: 0x150190000
Process17 size: 72 KB
------------------------------------------
Process18 starting address: 0x1501a2000
Process18 size: 92 KB
------------------------------------------
Process19 starting address: 0x1501b9000
Process19 size: 64 KB
------------------------------------------
Process20 starting address: 0x1501c9000
Process20 size: 60 KB
------------------------------------------
Process21 starting address: 0x1501d8000
Process21 size: 76 KB
------------------------------------------
Process22 starting address: 0x1501eb000
Process22 size: 32 KB
------------------------------------------

+++++++++++Hole Information+++++++++++
--------------------------------------
Hole0 starting address: 0x1501f3000
Hole0 size: 84 KB
--------------------------------------
```

```
===============Memory Information of Worst Fit algorithm===============
size: 1MB
Memory starting address: 0x150208000
Memory ending address:   0x150307fff

++++++++++Process Information++++++++++
------------------------------------------
Process0 starting address: 0x150208000
Process0 size: 16 KB
------------------------------------------
Process2 starting address: 0x15020c000
Process2 size: 4 KB
------------------------------------------
Process3 starting address: 0x15020d000
Process3 size: 72 KB
------------------------------------------
Process4 starting address: 0x15021f000
Process4 size: 72 KB
------------------------------------------
Process5 starting address: 0x150231000
Process5 size: 48 KB
------------------------------------------
Process6 starting address: 0x15023d000
Process6 size: 4 KB
------------------------------------------
Process7 starting address: 0x15023e000
Process7 size: 12 KB
------------------------------------------
Process8 starting address: 0x150241000
Process8 size: 76 KB
------------------------------------------
Process9 starting address: 0x150254000
Process9 size: 40 KB
------------------------------------------
Process10 starting address: 0x15025e000
Process10 size: 8 KB
------------------------------------------
Process11 starting address: 0x150260000
Process11 size: 28 KB
------------------------------------------
Process13 starting address: 0x150267000
Process13 size: 84 KB
------------------------------------------
Process14 starting address: 0x15027c000
Process14 size: 24 KB
------------------------------------------
Process15 starting address: 0x150282000
Process15 size: 48 KB
------------------------------------------
```

```
Process16 starting address: 0x15028e000
Process16 size: 8 KB
-----------------------------------------------
Process17 starting address: 0x150290000
Process17 size: 72 KB
-----------------------------------------------
Process18 starting address: 0x1502a2000
Process18 size: 92 KB
-----------------------------------------------
Process19 starting address: 0x1502b9000
Process19 size: 64 KB
-----------------------------------------------
Process20 starting address: 0x1502c9000
Process20 size: 60 KB
-----------------------------------------------
Process21 starting address: 0x1502d8000
Process21 size: 76 KB
-----------------------------------------------
Process22 starting address: 0x1502eb000
Process22 size: 32 KB
-----------------------------------------------

+++++++++++Hole Information+++++++++++
-------------------------------------------
Hole0 starting address: 0x1502f3000
Hole0 size: 84 KB
-------------------------------------------


==========================Memory Information after refilling==========================

================Memory Information of First Fit algorithm================
size: 1MB
Memory starting address: 0x150008000
Memory ending address:   0x150107fff

++++++++++Process Information++++++++++
-----------------------------------------------
Process0 starting address: 0x150008000
Process0 size: 16 KB
-----------------------------------------------
Process2 starting address: 0x15000c000
Process2 size: 4 KB
-----------------------------------------------
Process3 starting address: 0x15000d000
Process3 size: 72 KB
-----------------------------------------------
Process4 starting address: 0x15001f000
Process4 size: 72 KB
-----------------------------------------------
```

```
Process5 starting address: 0x150031000
Process5 size: 48 KB
```

```
Process6 starting address: 0x15003d000
Process6 size: 4 KB
```

```
Process7 starting address: 0x15003e000
Process7 size: 12 KB
```

```
Process8 starting address: 0x150041000
Process8 size: 76 KB
```

```
Process9 starting address: 0x150054000
Process9 size: 40 KB
```

```
Process10 starting address: 0x15005e000
Process10 size: 8 KB
```

```
Process11 starting address: 0x150060000
Process11 size: 28 KB
```

```
Process13 starting address: 0x150067000
Process13 size: 84 KB
```

```
Process14 starting address: 0x15007c000
Process14 size: 24 KB
```

```
Process15 starting address: 0x150082000
Process15 size: 48 KB
```

```
Process16 starting address: 0x15008e000
Process16 size: 8 KB
```

```
Process17 starting address: 0x150090000
Process17 size: 72 KB
```

```
Process18 starting address: 0x1500a2000
Process18 size: 92 KB
```

```
Process19 starting address: 0x1500b9000
Process19 size: 64 KB
```

```
Process20 starting address: 0x1500c9000
Process20 size: 60 KB
```

```
Process21 starting address: 0x1500d8000
Process21 size: 76 KB
```

```
Process22 starting address: 0x1500eb000
Process22 size: 32 KB
```

```
Process24 starting address: 0x1500f3000
Process24 size: 16 KB
-----------------------------------------
Process25 starting address: 0x1500f7000
Process25 size: 52 KB
-----------------------------------------
Process26 starting address: 0x150104000
Process26 size: 4 KB
-----------------------------------------
Process30 starting address: 0x150105000
Process30 size: 4 KB
-----------------------------------------
Process34 starting address: 0x150106000
Process34 size: 8 KB
-----------------------------------------

+++++++++++Hole Information+++++++++++
----------------------------------------

=================Memory Information of Best Fit algorithm=================
size: 1MB
Memory starting address: 0x150108000
Memory ending address:   0x150207fff

++++++++++Process Information++++++++++
-----------------------------------------
Process0 starting address: 0x150108000
Process0 size: 16 KB
-----------------------------------------
Process2 starting address: 0x15010c000
Process2 size: 4 KB
-----------------------------------------
Process3 starting address: 0x15010d000
Process3 size: 72 KB
-----------------------------------------
Process4 starting address: 0x15011f000
Process4 size: 72 KB
-----------------------------------------
Process5 starting address: 0x150131000
Process5 size: 48 KB
-----------------------------------------
Process6 starting address: 0x15013d000
Process6 size: 4 KB
-----------------------------------------
Process7 starting address: 0x15013e000
Process7 size: 12 KB
-----------------------------------------
Process8 starting address: 0x150141000
Process8 size: 76 KB
```

```
Process9 starting address: 0x150154000
Process9 size: 40 KB
```

```
Process10 starting address: 0x15015e000
Process10 size: 8 KB
```

```
Process11 starting address: 0x150160000
Process11 size: 28 KB
```

```
Process13 starting address: 0x150167000
Process13 size: 84 KB
```

```
Process14 starting address: 0x15017c000
Process14 size: 24 KB
```

```
Process15 starting address: 0x150182000
Process15 size: 48 KB
```

```
Process16 starting address: 0x15018e000
Process16 size: 8 KB
```

```
Process17 starting address: 0x150190000
Process17 size: 72 KB
```

```
Process18 starting address: 0x1501a2000
Process18 size: 92 KB
```

```
Process19 starting address: 0x1501b9000
Process19 size: 64 KB
```

```
Process20 starting address: 0x1501c9000
Process20 size: 60 KB
```

```
Process21 starting address: 0x1501d8000
Process21 size: 76 KB
```

```
Process22 starting address: 0x1501eb000
Process22 size: 32 KB
```

```
Process24 starting address: 0x1501f3000
Process24 size: 16 KB
```

```
Process25 starting address: 0x1501f7000
Process25 size: 52 KB
```

```
Process26 starting address: 0x150204000
Process26 size: 4 KB
```

```
Process30 starting address: 0x150205000
```

```
Process30 size: 4 KB
-----------------------------------------------
Process34 starting address: 0x150206000
Process34 size: 8 KB
-----------------------------------------------

+++++++++++Hole Information+++++++++++
-------------------------------------------

================Memory Information of Worst Fit algorithm================
size: 1MB
Memory starting address: 0x150208000
Memory ending address:   0x150307fff

++++++++++Process Information++++++++++
-----------------------------------------------
Process0 starting address: 0x150208000
Process0 size: 16 KB
-----------------------------------------------
Process2 starting address: 0x15020c000
Process2 size: 4 KB
-----------------------------------------------
Process3 starting address: 0x15020d000
Process3 size: 72 KB
-----------------------------------------------
Process4 starting address: 0x15021f000
Process4 size: 72 KB
-----------------------------------------------
Process5 starting address: 0x150231000
Process5 size: 48 KB
-----------------------------------------------
Process6 starting address: 0x15023d000
Process6 size: 4 KB
-----------------------------------------------
Process7 starting address: 0x15023e000
Process7 size: 12 KB
-----------------------------------------------
Process8 starting address: 0x150241000
Process8 size: 76 KB
-----------------------------------------------
Process9 starting address: 0x150254000
Process9 size: 40 KB
-----------------------------------------------
Process10 starting address: 0x15025e000
Process10 size: 8 KB
-----------------------------------------------
Process11 starting address: 0x150260000
Process11 size: 28 KB
-----------------------------------------------
Process13 starting address: 0x150267000
```

```
Process13 size: 84 KB
--------------------------------------------
Process14 starting address: 0x15027c000
Process14 size: 24 KB
--------------------------------------------
Process15 starting address: 0x150282000
Process15 size: 48 KB
--------------------------------------------
Process16 starting address: 0x15028e000
Process16 size: 8 KB
--------------------------------------------
Process17 starting address: 0x150290000
Process17 size: 72 KB
--------------------------------------------
Process18 starting address: 0x1502a2000
Process18 size: 92 KB
--------------------------------------------
Process19 starting address: 0x1502b9000
Process19 size: 64 KB
--------------------------------------------
Process20 starting address: 0x1502c9000
Process20 size: 60 KB
--------------------------------------------
Process21 starting address: 0x1502d8000
Process21 size: 76 KB
--------------------------------------------
Process22 starting address: 0x1502eb000
Process22 size: 32 KB
--------------------------------------------
Process24 starting address: 0x1502f3000
Process24 size: 16 KB
--------------------------------------------
Process25 starting address: 0x1502f7000
Process25 size: 52 KB
--------------------------------------------
Process26 starting address: 0x150304000
Process26 size: 4 KB
--------------------------------------------
Process30 starting address: 0x150305000
Process30 size: 4 KB
--------------------------------------------
Process34 starting address: 0x150306000
Process34 size: 8 KB
--------------------------------------------

+++++++++++Hole Information+++++++++++
--------------------------------------------
```

3 Code

3.1 Three algorithms without Compaction

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#define MEM_SIZE 1048576 //define the memory size(unit: byte)


/* algo 1-->first fit   2-->best fit  3-->worst fit */

/* Possible sizes of different processes */
int possible_sizes[] = {4, 8, 12, 16, 20,
                        24, 28, 32, 36, 40,
                        44, 48, 52, 56, 60,
                        64, 68, 72, 76, 80,
                        84, 88, 92, 96, 100};


/* record the starting addresses of different processes for three algorithms */
char* ff_pro_add[500];
char* bf_pro_add[500];
char* wf_pro_add[500];

/* record the sizes of different processes for three algorithms
   unit: KB */
int ff_pro_size[500];
int bf_pro_size[500];
int wf_pro_size[500];

/* process index of three algorithms */
int ff_pro_index = 0;
int bf_pro_index = 0;
int wf_pro_index = 0;

/* hole address information */
char* ff_hole_add[500];
char* bf_hole_add[500];
char* wf_hole_add[500];

/* hole size information
   unit: byte */
int ff_hole_size[500];
int bf_hole_size[500];
int wf_hole_size[500];

/* function declеartion */
int fill_up(int pro_num, int pro_size, char* mem_add, int mem_type);
int size_left(char* mem_add);

/*
pro_number: process number
pro_size  : process size(KB)
mem_add   : memory starting address
algo      : 1 --> ff memory ; 2 --> bf memory ; 3 --> wf memory
return 1 ---> fill successful
return 0 ---> fill unsuccessful
*/
int fill_up(int pro_num, int pro_size, char* mem_add, int algo)
{
    if(algo == 1)
    {
        if (size_left(mem_add) < pro_size * 1024)
        {
            return 0;
        }

        char* temp = mem_add;

        while(*temp == 'f')
```

```
        {
            temp++;
        }

        ff_pro_add[pro_num] = temp;
        ff_pro_size[pro_num] = pro_size;

        /* filling process */
        int i;
        for (i = 0; i < pro_size * 1024; i++)
        {
            *temp = 'f'; //f means filled up
            temp++;
        }

        return 1;
    }
    else if (algo == 2)
    {
        if (size_left(mem_add) < pro_size * 1024)
        {
            return 0;
        }

        char* temp = mem_add;

        while(*temp == 'f')
        {
            temp++;
        }

        bf_pro_add[pro_num] = temp;
        bf_pro_size[pro_num] = pro_size;

        /* filling process */
        int i;
        for (i = 0; i < pro_size * 1024; i++)
        {
            *temp = 'f'; //f means filled up
            temp++;
        }

        return 1;
    }
    else
    {
        if (size_left(mem_add) < pro_size * 1024)
        {
            return 0;
        }

        char* temp = mem_add;

        while(*temp == 'f')
        {
            temp++;
        }

        wf_pro_add[pro_num] = temp;
        wf_pro_size[pro_num] = pro_size;

        /* filling process */
        int i;
        for (i = 0; i < pro_size * 1024; i++)
        {
            *temp = 'f'; //f means filled up
            temp++;
        }

        return 1;
    }
}
```

```
/* calculate the size left for a memory space
   return unit: byte */
int size_left(char* mem_add)
{
    int i;
    int counter = 0;
    for(i = 0; i < MEM_SIZE; i++)
    {
        if(*(mem_add + i) == 'e')
        {
            counter++;
        }
    }
    return counter;
}

/* algo 1-->first fit; 2-->best fit  3-->worst fit */
int number_of_processes(int algo)
{
    int counter = 0;
    int i;
    if(algo == 1)
    {
        for(i = 0; i < 500; i++)
        {
            if (ff_pro_size[i] != 0)
            {
                counter++;
            }
        }
    }
    else if (algo == 2)
    {
        for(i = 0; i < 500; i++)
        {
            if(bf_pro_size[i] != 0)
            {
                counter++;
            }
        }
    }
    else
    {
        for(i = 0; i < 500; i++)
        {
            if(wf_pro_size[i] != 0)
            {
                counter++;
            }
        }
    }
    return counter;
}

void ff_get_hole_information(char* mem_loc)
{
    char* mem_end = mem_loc + (1024 * 1024) - 1;
    int i;
    for (i = 0; i < 500; i++)
    {
        ff_hole_add[i] = NULL;
        ff_hole_size[i] = 0;
    }
    i = 0;
    char* start = mem_loc;
    char* end = mem_loc;
    while(1)
    {
        if(*start == 'f' && *end == 'f')
        {
            if (start == mem_end && end == mem_end)
```

```
            {
                return;
            }
            start++;
            end++;
        }
        else
        {

            if(start == mem_end && end == mem_end)
            {
                ff_hole_add[i] = start;
                ff_hole_size[i] = 1;
                return;
            }

            while(*end == 'e')
            {
                if (end == mem_end)
                {
                    ff_hole_add[i] = start;
                    ff_hole_size[i] = end - start + 1;
                    return;
                }
                end++;
            }
            ff_hole_add[i] = start;
            ff_hole_size[i] = end - start;
            start = end;
            i++;
        }
    }
}

void bf_get_hole_information(char* mem_loc)
{
    char* mem_end = mem_loc + (1024 * 1024) - 1;
    int i;
    for (i = 0; i < 500; i++)
    {
        bf_hole_add[i] = NULL;
        bf_hole_size[i] = 0;
    }
    i = 0;
    char* start = mem_loc;
    char* end = mem_loc;
    while(1)
    {
        if(*start == 'f' && *end == 'f')
        {
            if (start == mem_end && end == mem_end)
            {
                return;
            }
            start++;
            end++;
        }
        else
        {

            if(start == mem_end && end == mem_end)
            {
                bf_hole_add[i] = start;
                bf_hole_size[i] = 1;
                return;
            }

            while(*end == 'e')
            {
                if (end == mem_end)
                {
                    bf_hole_add[i] = start;
```

```
                    bf_hole_size[i] = end - start + 1;
                    return;
                }
                end++;
            }
            bf_hole_add[i] = start;
            bf_hole_size[i] = end - start;
            start = end;
            i++;
        }
    }
}

void wf_get_hole_information(char* mem_loc)
{
    char* mem_end = mem_loc + (1024 * 1024) - 1;
    int i;
    for (i = 0; i < 500; i++)
    {
        wf_hole_add[i] = NULL;
        wf_hole_size[i] = 0;
    }
    i = 0;
    char* start = mem_loc;
    char* end = mem_loc;
    while(1)
    {
        if(*start == 'f' && *end == 'f')
        {
            if (start == mem_end && end == mem_end)
            {
                return;
            }
            start++;
            end++;
        }
        else
        {

            if(start == mem_end && end == mem_end)
            {
                wf_hole_add[i] = start;
                wf_hole_size[i] = 1;
                return;
            }

            while(*end == 'e')
            {
                if (end == mem_end)
                {
                    wf_hole_add[i] = start;
                    wf_hole_size[i] = end - start + 1;
                    return;
                }
                end++;
            }
            wf_hole_add[i] = start;
            wf_hole_size[i] = end - start;
            start = end;
            i++;
        }
    }
}

void update_hole_info(char* mem_add, int algo)
{
    if(algo == 1)
    {
        ff_get_hole_information(mem_add);
    }
    else if (algo == 2)
    {
```

```
        bf_get_hole_information(mem_add);
    }
    else
    {
        wf_get_hole_information(mem_add);
    }
}

int number_of_holes(int algo)
{
    int i;
    int counter = 0;
    if(algo == 1)
    {
        for(i = 0; i < 500; i++)
        {
            if(ff_hole_size[i] != 0)
            {
                counter++;
            }
        }
    }
    else if(algo == 2)
    {
        for(i = 0; i < 500; i++)
        {
            if(bf_hole_size[i] != 0)
            {
                counter++;
            }
        }
    }
    else
    {
        for(i = 0; i < 500; i++)
        {
            if(wf_hole_size[i] != 0)
            {
                counter++;
            }
        }
    }
    return counter;
}

/* return 1: allocation successful
   return 0: allocation fail
   process_size unit : KB */
int allocate_ff(int process_number, int process_size, char* mem_loc)
{
    /* update hole information */
    update_hole_info(mem_loc, 1);
    int hole_num = number_of_holes(1);
    int i;
    int j;
    for (int i = 0; i < hole_num; i++)
    {
        if(ff_hole_size[i] >= process_size * 1024)
        {
            ff_pro_add[process_number] = ff_hole_add[i];
            ff_pro_size[process_number] = process_size;
            for(j = 0; j < process_size * 1024; j++)
            {
                *(ff_hole_add[i] + j) = 'f';
            }
            return 1;
        }
    }
    return 0;
}

/* return 1: allocation successful
```

```
   return 0: allocation fail
   process_size unit : KB
   Best fit approach*/
int allocate_bf(int process_number, int process_size, char* mem_loc)
{
    int i;
    int j;

    /* update hole information */
    update_hole_info(mem_loc, 2);
    int hole_num = number_of_holes(2);

    /* No holes exist */
    if (hole_num == 0)
    {
        return 0;
    }

    int difference[hole_num];   //for each cell: calculate hole_size[i] - process_size
    int smallestPositveDiffIndex = -1;

    /* calculate the difference */
    for(i = 0; i < hole_num; i++)
    {
        difference[i] = bf_hole_size[i] - process_size * 1024;
    }

    /* check if there is any positve difference */
    for(i = 0; i < hole_num; i++)
    {
        if (difference[i] >= 0)
        {
            smallestPositveDiffIndex = i;
        }
    }

    /* All holes are too small */
    if(smallestPositveDiffIndex == -1)
    {
        return 0;
    }

    /* find the hole with the smallest difference */
    for(i = 0; i < hole_num; i++)
    {
        if (difference[i] >= 0 && difference[i] < difference[smallestPositveDiffIndex])
        {
            smallestPositveDiffIndex = i;
        }
    }


    bf_pro_add[process_number] = bf_hole_add[smallestPositveDiffIndex];
    bf_pro_size[process_number] = process_size;
    for(j = 0; j < process_size * 1024; j++)
    {
        *(bf_hole_add[smallestPositveDiffIndex] + j) = 'f';
    }
    return 1;
}

/* return 1: allocation successful
   return 0: allocation fail
   process_size unit : KB
   Worst fit approach*/
int allocate_wf(int process_number, int process_size, char* mem_loc)
{
    int i;
    int j;

    /* update hole information */
    update_hole_info(mem_loc, 3);
```

```
    int hole_num = number_of_holes(3);

    /* No holes exist */
    if (hole_num == 0)
    {
        return 0;
    }

    int difference[hole_num];    //for each cell: calculate hole_size[i] - process_size
    int largestPositveDiffIndex = -1;

    /* calculate the difference */
    for(i = 0; i < hole_num; i++)
    {
        difference[i] = wf_hole_size[i] - process_size * 1024;
    }

    /* check if there is any positve difference */
    for(i = 0; i < hole_num; i++)
    {
        if (difference[i] >= 0)
        {
            largestPositveDiffIndex = i;
        }
    }

    /* All holes are too small */
    if(largestPositveDiffIndex == -1)
    {
        return 0;
    }

    /* find the hole with the biggest difference */
    for(i = 0; i < hole_num; i++)
    {
        if (difference[i] >= 0 && difference[i] > difference[largestPositveDiffIndex])
        {
            largestPositveDiffIndex = i;
        }
    }


    wf_pro_add[process_number] = wf_hole_add[largestPositveDiffIndex];
    wf_pro_size[process_number] = process_size;
    for(j = 0; j < process_size * 1024; j++)
    {
        *(wf_hole_add[largestPositveDiffIndex] + j) = 'f';
    }
    return 1;
}


/*
process_size (unit:KB)
return1: allocate successfully
return0: allocation failed
*/
int allocate(int process_number, int process_size, char* mem_loc, int algo)
{
    int indicator;
    if(algo == 1)
    {
        indicator = allocate_ff(process_number, process_size, mem_loc);
    }
    else if(algo == 2)
    {
        indicator = allocate_bf(process_number, process_size, mem_loc);
    }
    else
    {
        indicator = allocate_wf(process_number, process_size, mem_loc);
    }
```

```c
    return indicator;
}

/* algo: 1--> ff 2-->bf  3-->wf */
void status(char* mem_add, int algo)
{
    int i;
    if(algo == 1)
    {
        printf("===============Memory Information of First Fit algorithm===============\n");
        printf("size: 1MB\n");
        printf("Memory starting address: %p\n", mem_add);
        printf("Memory ending address:   %p\n", mem_add + (1024 * 1024) - 1);
        printf("\n");
        printf("++++++++++Process Information++++++++++\n");
        printf("----------------------------------------\n");
        for(int i = 0; i < 500; i++)
        {
            if (ff_pro_size[i] != 0)
            {
                printf("Process%d starting address: %p\n", i, ff_pro_add[i]);
                printf("Process%d size: %d KB\n", i, ff_pro_size[i]);
                printf("----------------------------------------\n");
            }
        }
        printf("\n");
        printf("+++++++++++Hole Information+++++++++++\n");
        printf("--------------------------------------\n");
        for(i = 0; i < 500; i++)
        {
            if(ff_hole_size[i] != 0)
            {
                printf("Hole%d starting address: %p\n", i, ff_hole_add[i]);
                printf("Hole%d size: %d KB\n", i, ff_hole_size[i] / 1024);
                printf("--------------------------------------\n");
            }
        }
        printf("\n");
    }
    else if (algo == 2)
    {
        printf("===============Memory Information of Best Fit algorithm===============\n");
        printf("size: 1MB\n");
        printf("Memory starting address: %p\n", mem_add);
        printf("Memory ending address:   %p\n", mem_add + (1024 * 1024) - 1);
        printf("\n");
        printf("++++++++++Process Information++++++++++\n");
        printf("----------------------------------------\n");
        for(int i = 0; i < 500; i++)
        {
            if (bf_pro_size[i] != 0)
            {
                printf("Process%d starting address: %p\n", i, bf_pro_add[i]);
                printf("Process%d size: %d KB\n", i, bf_pro_size[i]);
                printf("----------------------------------------\n");
            }
        }
        printf("\n");
        printf("+++++++++++Hole Information+++++++++++\n");
        printf("--------------------------------------\n");
        for(i = 0; i < 500; i++)
        {
            if(bf_hole_size[i] != 0)
            {
                printf("Hole%d starting address: %p\n", i, bf_hole_add[i]);
                printf("Hole%d size: %d KB\n", i, bf_hole_size[i] / 1024);
                printf("--------------------------------------\n");
            }
        }
        printf("\n");
    }
    else
```

```c
    {
        printf("==================Memory Information of Worst Fit algorithm===================\n");
        printf("size: 1MB\n");
        printf("Memory starting address: %p\n", mem_add);
        printf("Memory ending address:   %p\n", mem_add + (1024 * 1024) - 1);
        printf("\n");
        printf("++++++++++Process Information++++++++++\n");
        printf("----------------------------------------\n");
        for(int i = 0; i < 500; i++)
        {
            if (wf_pro_size[i] != 0)
            {
                printf("Process%d starting address: %p\n", i, wf_pro_add[i]);
                printf("Process%d size: %d KB\n", i, wf_pro_size[i]);
                printf("----------------------------------------\n");
            }
        }
        printf("\n");
        printf("+++++++++++Hole Information+++++++++++\n");
        printf("--------------------------------------\n");
        for(i = 0; i < 500; i++)
        {
            if(wf_hole_size[i] != 0)
            {
                printf("Hole%d starting address: %p\n", i, wf_hole_add[i]);
                printf("Hole%d size: %d KB\n", i, wf_hole_size[i] / 1024);
                printf("--------------------------------------\n");
            }
        }
        printf("\n");
    }
}

/* algo 1-->frist fit   2-->best fit  3-->worst fit
   return 0 means that process has aleady been released
   return 1 means that process can be released */
int release(int pro_num, int algo)
{
    if(algo == 1)
    {
        /* process has been released already */
        if (ff_pro_add[pro_num] == 0)
        {
            return 0;
        }
        int byte_size = ff_pro_size[pro_num] * 1024;
        int i;
        for(i = 0; i < byte_size; i++)
        {
            *(ff_pro_add[pro_num] + i) = 'e';
        }
        ff_pro_size[pro_num] = 0;
        ff_pro_add[pro_num] = NULL;
        return 1;
    }
    else if (algo == 2)
    {
        /* process has been released already */
        if (bf_pro_add[pro_num] == 0)
        {
            return 0;
        }
        int byte_size = bf_pro_size[pro_num] * 1024;
        int i;
        for(i = 0; i < byte_size; i++)
        {
            *(bf_pro_add[pro_num] + i) = 'e';
        }
        bf_pro_size[pro_num] = 0;
        bf_pro_add[pro_num] = NULL;
        return 1;
    }
```

```c
    else
    {
        /* process has been released already */
        if (wf_pro_add[pro_num] == 0)
        {
            return 0;
        }
        int byte_size = wf_pro_size[pro_num] * 1024;
        int i;
        for(i = 0; i < byte_size; i++)
        {
            *(wf_pro_add[pro_num] + i) = 'e';
        }
        wf_pro_size[pro_num] = 0;
        wf_pro_add[pro_num] = NULL;
        return 1;
    }
}

int main()
{
    int i;
    int random;
    int stop_indicator;

    /* initialize three block of memories to implement three algoritms */
    char* ff_mem_add = (char*)malloc(MEM_SIZE);
    char* bf_mem_add = (char*)malloc(MEM_SIZE);
    char* wf_mem_add = (char*)malloc(MEM_SIZE);
    for (i = 0; i < 1024 * 1024; i++)
    {
        *(ff_mem_add + i) = 'e'; //e means empty
        *(bf_mem_add + i) = 'e'; //e means empty
        *(wf_mem_add + i) = 'e'; //e means empty
    }

    /* filling process */
    srand(time(NULL));
    while(1)
    {
        random = rand() % 25;  //generate random number [0, 24]
        stop_indicator = fill_up(ff_pro_index, possible_sizes[random], ff_mem_add, 1);
        fill_up(bf_pro_index, possible_sizes[random], bf_mem_add, 2);
        fill_up(wf_pro_index, possible_sizes[random], wf_mem_add, 3);
        ff_pro_index++;
        bf_pro_index++;
        wf_pro_index++;
        if(stop_indicator == 0)
        {
            break;
        }
    }


    /* randomly release 10% */
    int total_process_number = number_of_processes(1);
    int number_needed_to_remove = total_process_number / 10;
    int temp = total_process_number % 10;
    int isremoved;
    srand(time(NULL));
    if(temp >= 5)
    {
        number_needed_to_remove++;
    }
    while(1)
    {
        random = rand() % total_process_number;
        isremoved = release(random, 1);
        release(random, 2);
        release(random, 3);
        if(isremoved == 1)
        {
```

```
            number_needed_to_remove--;
        }
        if(number_needed_to_remove == 0)
        {
            break;
        }
    }

    /* update hole information after removing 10% */
    update_hole_info(ff_mem_add, 1);
    update_hole_info(bf_mem_add, 2);
    update_hole_info(wf_mem_add, 3);

    /* Display memory information after removing 10% */
    printf("==================Memory Information after removing 10 percent====================\n");
    printf("\n");
    status(ff_mem_add, 1);
    status(bf_mem_add, 2);
    status(wf_mem_add, 3);

    /* refilling the memories */
    int ff_stop_indicator = 1;
    int bf_stop_indicator = 1;
    int wf_stop_indicator = 1;
    srand(time(NULL));
    i = 0;
    while(i <= 10)
    {
        random = rand() % 25;
        allocate(ff_pro_index, possible_sizes[random], ff_mem_add, 1);
        allocate(bf_pro_index, possible_sizes[random], bf_mem_add, 2);
        allocate(wf_pro_index, possible_sizes[random], wf_mem_add, 3);
        i++;
        ff_pro_index++;
        bf_pro_index++;
        wf_pro_index++;
    }

    /* update hole information afte refilling */
    update_hole_info(ff_mem_add, 1);
    update_hole_info(bf_mem_add, 2);
    update_hole_info(wf_mem_add, 3);

    /* Display memory information after refilling */
    printf("======================Memory Information after refilling========================\n");
    printf("\n");
    status(ff_mem_add, 1);
    status(bf_mem_add, 2);
    status(wf_mem_add, 3);

    return 0;
}
```

3.2 Three algorithms with Compaction

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#define MEM_SIZE 1048576 //define the memory size(unit: byte)


/* algo 1-->first fit   2-->best fit  3-->worst fit */

/* Possible sizes of different processes */
int possible_sizes[] = {4, 8, 12, 16, 20,
                        24, 28, 32, 36, 40,
                        44, 48, 52, 56, 60,
                        64, 68, 72, 76, 80,
                        84, 88, 92, 96, 100};


/* record the starting addresses of different processes for three algorithms */
char* ff_pro_add[500];
char* bf_pro_add[500];
char* wf_pro_add[500];

/* record the sizes of different processes for three algorithms
   unit: KB */
int ff_pro_size[500];
int bf_pro_size[500];
int wf_pro_size[500];

/* process index of three algorithms */
int ff_pro_index = 0;
int bf_pro_index = 0;
int wf_pro_index = 0;

/* hole address information */
char* ff_hole_add[500];
char* bf_hole_add[500];
char* wf_hole_add[500];

/* hole size information
   unit: byte */
int ff_hole_size[500];
int bf_hole_size[500];
int wf_hole_size[500];

/* function decleartion */
int fill_up(int pro_num, int pro_size, char* mem_add, int mem_type);
int size_left(char* mem_add);

/*
pro_number: process number
pro_size  : process size(KB)
mem_add   : memory starting address
algo      : 1 --> ff memory ; 2 --> bf memory ; 3 --> wf memory
return 1 ---> fill successful
return 0 ---> fill unsuccessful
*/
int fill_up(int pro_num, int pro_size, char* mem_add, int algo)
{
    if(algo == 1)
    {
        if (size_left(mem_add) < pro_size * 1024)
        {
            return 0;
        }

        char* temp = mem_add;

        while(*temp == 'f')
        {
            temp++;
        }
```

```c
        ff_pro_add[pro_num] = temp;
        ff_pro_size[pro_num] = pro_size;

        /* filling process */
        int i;
        for (i = 0; i < pro_size * 1024; i++)
        {
            *temp = 'f'; //f means filled up
            temp++;
        }

        return 1;
    }
    else if (algo == 2)
    {
        if (size_left(mem_add) < pro_size * 1024)
        {
            return 0;
        }

        char* temp = mem_add;

        while(*temp == 'f')
        {
            temp++;
        }

        bf_pro_add[pro_num] = temp;
        bf_pro_size[pro_num] = pro_size;

        /* filling process */
        int i;
        for (i = 0; i < pro_size * 1024; i++)
        {
            *temp = 'f'; //f means filled up
            temp++;
        }

        return 1;
    }
    else
    {
        if (size_left(mem_add) < pro_size * 1024)
        {
            return 0;
        }

        char* temp = mem_add;

        while(*temp == 'f')
        {
            temp++;
        }

        wf_pro_add[pro_num] = temp;
        wf_pro_size[pro_num] = pro_size;

        /* filling process */
        int i;
        for (i = 0; i < pro_size * 1024; i++)
        {
            *temp = 'f'; //f means filled up
            temp++;
        }

        return 1;
    }
}

/* calculate the size left for a memory space
   return unit: byte */
```

```c
int size_left(char* mem_add)
{
    int i;
    int counter = 0;
    for(i = 0; i < MEM_SIZE; i++)
    {
        if(*(mem_add + i) == 'e')
        {
            counter++;
        }
    }
    return counter;
}

/* algo 1-->first fit; 2-->best fit 3-->worst fit */
int number_of_processes(int algo)
{
    int counter = 0;
    int i;
    if(algo == 1)
    {
        for(i = 0; i < 500; i++)
        {
            if (ff_pro_size[i] != 0)
            {
                counter++;
            }
        }
    }
    else if (algo == 2)
    {
        for(i = 0; i < 500; i++)
        {
            if(bf_pro_size[i] != 0)
            {
                counter++;
            }
        }
    }
    else
    {
        for(i = 0; i < 500; i++)
        {
            if(wf_pro_size[i] != 0)
            {
                counter++;
            }
        }
    }
    return counter;
}

void ff_get_hole_information(char* mem_loc)
{
    char* mem_end = mem_loc + (1024 * 1024) - 1;
    int i;
    for (i = 0; i < 500; i++)
    {
        ff_hole_add[i] = NULL;
        ff_hole_size[i] = 0;
    }
    i = 0;
    char* start = mem_loc;
    char* end = mem_loc;
    while(1)
    {
        if(*start == 'f' && *end == 'f')
        {
            if (start == mem_end && end == mem_end)
            {
                return;
            }
```

```
                start++;
                end++;
            }
            else
            {

                if(start == mem_end && end == mem_end)
                {
                    ff_hole_add[i] = start;
                    ff_hole_size[i] = 1;
                    return;
                }

                while(*end == 'e')
                {
                    if (end == mem_end)
                    {
                        ff_hole_add[i] = start;
                        ff_hole_size[i] = end - start + 1;
                        return;
                    }
                    end++;
                }
                ff_hole_add[i] = start;
                ff_hole_size[i] = end - start;
                start = end;
                i++;
            }
        }
}

void bf_get_hole_information(char* mem_loc)
{
    char* mem_end = mem_loc + (1024 * 1024) - 1;
    int i;
    for (i = 0; i < 500; i++)
    {
        bf_hole_add[i] = NULL;
        bf_hole_size[i] = 0;
    }
    i = 0;
    char* start = mem_loc;
    char* end = mem_loc;
    while(1)
    {
        if(*start == 'f' && *end == 'f')
        {
            if (start == mem_end && end == mem_end)
            {
                return;
            }
            start++;
            end++;
        }
        else
        {

            if(start == mem_end && end == mem_end)
            {
                bf_hole_add[i] = start;
                bf_hole_size[i] = 1;
                return;
            }

            while(*end == 'e')
            {
                if (end == mem_end)
                {
                    bf_hole_add[i] = start;
                    bf_hole_size[i] = end - start + 1;
                    return;
                }
```

```
                end++;
            }
            bf_hole_add[i] = start;
            bf_hole_size[i] = end - start;
            start = end;
            i++;
        }
    }
}

void wf_get_hole_information(char* mem_loc)
{
    char* mem_end = mem_loc + (1024 * 1024) - 1;
    int i;
    for (i = 0; i < 500; i++)
    {
        wf_hole_add[i] = NULL;
        wf_hole_size[i] = 0;
    }
    i = 0;
    char* start = mem_loc;
    char* end = mem_loc;
    while(1)
    {
        if(*start == 'f' && *end == 'f')
        {
            if (start == mem_end && end == mem_end)
            {
                return;
            }
            start++;
            end++;
        }
        else
        {

            if(start == mem_end && end == mem_end)
            {
                wf_hole_add[i] = start;
                wf_hole_size[i] = 1;
                return;
            }

            while(*end == 'e')
            {
                if (end == mem_end)
                {
                    wf_hole_add[i] = start;
                    wf_hole_size[i] = end - start + 1;
                    return;
                }
                end++;
            }
            wf_hole_add[i] = start;
            wf_hole_size[i] = end - start;
            start = end;
            i++;
        }
    }
}

void update_hole_info(char* mem_add, int algo)
{
    if(algo == 1)
    {
        ff_get_hole_information(mem_add);
    }
    else if (algo == 2)
    {
        bf_get_hole_information(mem_add);
    }
    else
```

```
    {
        wf_get_hole_information(mem_add);
    }
}

int number_of_holes(int algo)
{
    int i;
    int counter = 0;
    if(algo == 1)
    {
        for(i = 0; i < 500; i++)
        {
            if(ff_hole_size[i] != 0)
            {
                counter++;
            }
        }
    }
    else if(algo == 2)
    {
        for(i = 0; i < 500; i++)
        {
            if(bf_hole_size[i] != 0)
            {
                counter++;
            }
        }
    }
    else
    {
        for(i = 0; i < 500; i++)
        {
            if(wf_hole_size[i] != 0)
            {
                counter++;
            }
        }
    }
    return counter;
}

/* return 1: allocation successful
   return 0: allocation fail
   process_size unit : KB */
int allocate_ff(int process_number, int process_size, char* mem_loc)
{
    /* update hole information */
    update_hole_info(mem_loc, 1);
    int hole_num = number_of_holes(1);
    int i;
    int j;
    for (int i = 0; i < hole_num; i++)
    {
        if(ff_hole_size[i] >= process_size * 1024)
        {
            ff_pro_add[process_number] = ff_hole_add[i];
            ff_pro_size[process_number] = process_size;
            for(j = 0; j < process_size * 1024; j++)
            {
                *(ff_hole_add[i] + j) = 'f';
            }
            return 1;
        }
    }
    return 0;
}

/* return 1: allocation successful
   return 0: allocation fail
   process_size unit : KB
   Best fit approach*/
```

```
int allocate_bf(int process_number, int process_size, char* mem_loc)
{
    int i;
    int j;

    /* update hole information */
    update_hole_info(mem_loc, 2);
    int hole_num = number_of_holes(2);

    /* No holes exist */
    if (hole_num == 0)
    {
        return 0;
    }

    int difference[hole_num];   //for each cell: calculate hole_size[i] - process_size
    int smallestPositveDiffIndex = -1;

    /* calculate the difference */
    for(i = 0; i < hole_num; i++)
    {
        difference[i] = bf_hole_size[i] - process_size * 1024;
    }

    /* check if there is any positve difference */
    for(i = 0; i < hole_num; i++)
    {
        if (difference[i] >= 0)
        {
            smallestPositveDiffIndex = i;
        }
    }

    /* All holes are too small */
    if(smallestPositveDiffIndex == -1)
    {
        return 0;
    }

    /* find the hole with the smallest difference */
    for(i = 0; i < hole_num; i++)
    {
        if (difference[i] >= 0 && difference[i] < difference[smallestPositveDiffIndex])
        {
            smallestPositveDiffIndex = i;
        }
    }


    bf_pro_add[process_number] = bf_hole_add[smallestPositveDiffIndex];
    bf_pro_size[process_number] = process_size;
    for(j = 0; j < process_size * 1024; j++)
    {
        *(bf_hole_add[smallestPositveDiffIndex] + j) = 'f';
    }
    return 1;
}

/* return 1: allocation successful
   return 0: allocation fail
   process_size unit : KB
   Worst fit approach*/
int allocate_wf(int process_number, int process_size, char* mem_loc)
{
    int i;
    int j;

    /* update hole information */
    update_hole_info(mem_loc, 3);
    int hole_num = number_of_holes(3);

    /* No holes exist */
```

```
    if (hole_num == 0)
    {
        return 0;
    }

    int difference[hole_num];   //for each cell: calculate hole_size[i] - process_size
    int largestPositveDiffIndex = -1;

    /* calculate the difference */
    for(i = 0; i < hole_num; i++)
    {
        difference[i] = wf_hole_size[i] - process_size * 1024;
    }

    /* check if there is any positve difference */
    for(i = 0; i < hole_num; i++)
    {
        if (difference[i] >= 0)
        {
            largestPositveDiffIndex = i;
        }
    }

    /* All holes are too small */
    if(largestPositveDiffIndex == -1)
    {
        return 0;
    }

    /* find the hole with the biggest difference */
    for(i = 0; i < hole_num; i++)
    {
        if (difference[i] >= 0 && difference[i] > difference[largestPositveDiffIndex])
        {
            largestPositveDiffIndex = i;
        }
    }


    wf_pro_add[process_number] = wf_hole_add[largestPositveDiffIndex];
    wf_pro_size[process_number] = process_size;
    for(j = 0; j < process_size * 1024; j++)
    {
        *(wf_hole_add[largestPositveDiffIndex] + j) = 'f';
    }
    return 1;
}

void compact_ff(char* mem_loc)
{
    int i;
    int total_space_needed = 0;
    for(i = 0; i < 500; i++)
    {
        total_space_needed += ff_pro_size[i] * 1024;
    }

    /* modify first process */
    int first_process_index;
    for(i = 0; i < 500; i++)
    {
        if (ff_pro_size[i] != 0)
        {
            first_process_index = i;
            break;
        }
    }
    ff_pro_add[first_process_index] = mem_loc;
    int j = first_process_index;
    for (i = 0; i < 500; i++)
    {
        if (ff_pro_size[i] != 0 && i != j)
```

```
        {
            ff_pro_add[i] = ff_pro_add[j] + ff_pro_size[j] * 1024;
            j = i;
        }
    }

    /* empty first */
    for (i = 0; i < 1024 * 1024; i++)
    {
        *(mem_loc + i) = 'e'; //e means empty
    }

    for (i = 0; i < total_space_needed; i++)
    {
        *(mem_loc + i) = 'f'; //e means empty
    }
}

void compact_bf(char* mem_loc)
{
    int i;
    int total_space_needed = 0;
    for(i = 0; i < 500; i++)
    {
        total_space_needed += bf_pro_size[i] * 1024;
    }

    /* modify first process */
    int first_process_index;
    for(i = 0; i < 500; i++)
    {
        if (bf_pro_size[i] != 0)
        {
            first_process_index = i;
            break;
        }
    }
    bf_pro_add[first_process_index] = mem_loc;
    int j = first_process_index;
    for (i = 0; i < 500; i++)
    {
        if (bf_pro_size[i] != 0 && i != j)
        {
            bf_pro_add[i] = bf_pro_add[j] + bf_pro_size[j] * 1024;
            j = i;
        }
    }

    /* empty first */
    for (i = 0; i < 1024 * 1024; i++)
    {
        *(mem_loc + i) = 'e'; //e means empty
    }

    for (i = 0; i < total_space_needed; i++)
    {
        *(mem_loc + i) = 'f'; //e means empty
    }
}


void compact_wf(char* mem_loc)
{
    int i;
    int total_space_needed = 0;
    for(i = 0; i < 500; i++)
    {
        total_space_needed += wf_pro_size[i] * 1024;
    }

    /* modify first process */
    int first_process_index;
```

```c
    for(i = 0; i < 500; i++)
    {
        if (wf_pro_size[i] != 0)
        {
            first_process_index = i;
            break;
        }
    }
    wf_pro_add[first_process_index] = mem_loc;
    int j = first_process_index;
    for (i = 0; i < 500; i++)
    {
        if (wf_pro_size[i] != 0 && i != j)
        {
            wf_pro_add[i] = wf_pro_add[j] + wf_pro_size[j] * 1024;
            j = i;
        }
    }

    /* empty first */
    for (i = 0; i < 1024 * 1024; i++)
    {
        *(mem_loc + i) = 'e'; //e means empty
    }

    for (i = 0; i < total_space_needed; i++)
    {
        *(mem_loc + i) = 'f'; //e means empty
    }
}

void compact(char* mem_loc, int algo)
{
    if(algo == 1)
    {
        compact_ff(mem_loc);
    }
    else if(algo == 2)
    {
        compact_bf(mem_loc);
    }
    else
    {
        compact_wf(mem_loc);
    }
}

/*
process_size (unit:KB)
return1: allocate successfully
return0: allocation failed
*/
int allocate(int process_number, int process_size, char* mem_loc, int algo)
{
    int indicator;
    if(algo == 1)
    {
        indicator = allocate_ff(process_number, process_size, mem_loc);
    }
    else if(algo == 2)
    {
        indicator = allocate_bf(process_number, process_size, mem_loc);
    }
    else
    {
        indicator = allocate_wf(process_number, process_size, mem_loc);
    }
    return indicator;
}

/* algo: 1--> ff 2-->bf  3-->wf */
void status(char* mem_add, int algo)
```

```
{
    int i;
    if(algo == 1)
    {
        printf("===============Memory Information of First Fit algorithm===============\n");
        printf("size: 1MB\n");
        printf("Memory starting address: %p\n", mem_add);
        printf("Memory ending address:   %p\n", mem_add + (1024 * 1024) - 1);
        printf("\n");
        printf("++++++++++Process Information++++++++++\n");
        printf("--------------------------------------------------\n");
        for(int i = 0; i < 500; i++)
        {
            if (ff_pro_size[i] != 0)
            {
                printf("Process%d starting address: %p\n", i, ff_pro_add[i]);
                printf("Process%d size: %d KB\n", i, ff_pro_size[i]);
                printf("--------------------------------------------------\n");
            }
        }
        printf("\n");
        printf("+++++++++++Hole Information+++++++++++\n");
        printf("------------------------------------------------\n");
        for(i = 0; i < 500; i++)
        {
            if(ff_hole_size[i] != 0)
            {
                printf("Hole%d starting address: %p\n", i, ff_hole_add[i]);
                printf("Hole%d size: %d KB\n", i, ff_hole_size[i] / 1024);
                printf("------------------------------------------------\n");
            }
        }
        printf("\n");
    }
    else if (algo == 2)
    {
        printf("===============Memory Information of Best Fit algorithm===============\n");
        printf("size: 1MB\n");
        printf("Memory starting address: %p\n", mem_add);
        printf("Memory ending address:   %p\n", mem_add + (1024 * 1024) - 1);
        printf("\n");
        printf("++++++++++Process Information++++++++++\n");
        printf("--------------------------------------------------\n");
        for(int i = 0; i < 500; i++)
        {
            if (bf_pro_size[i] != 0)
            {
                printf("Process%d starting address: %p\n", i, bf_pro_add[i]);
                printf("Process%d size: %d KB\n", i, bf_pro_size[i]);
                printf("--------------------------------------------------\n");
            }
        }
        printf("\n");
        printf("+++++++++++Hole Information+++++++++++\n");
        printf("------------------------------------------------\n");
        for(i = 0; i < 500; i++)
        {
            if(bf_hole_size[i] != 0)
            {
                printf("Hole%d starting address: %p\n", i, bf_hole_add[i]);
                printf("Hole%d size: %d KB\n", i, bf_hole_size[i] / 1024);
                printf("------------------------------------------------\n");
            }
        }
        printf("\n");
    }
    else
    {
        printf("===============Memory Information of Worst Fit algorithm===============\n");
        printf("size: 1MB\n");
        printf("Memory starting address: %p\n", mem_add);
        printf("Memory ending address:   %p\n", mem_add + (1024 * 1024) - 1);
```

```c
        printf("\n");
        printf("++++++++++Process Information++++++++++\n");
        printf("----------------------------------------\n");
        for(int i = 0; i < 500; i++)
        {
            if (wf_pro_size[i] != 0)
            {
                printf("Process%d starting address: %p\n", i, wf_pro_add[i]);
                printf("Process%d size: %d KB\n", i, wf_pro_size[i]);
                printf("----------------------------------------\n");
            }
        }
        printf("\n");
        printf("+++++++++++Hole Information+++++++++++\n");
        printf("----------------------------------------\n");
        for(i = 0; i < 500; i++)
        {
            if(wf_hole_size[i] != 0)
            {
                printf("Hole%d starting address: %p\n", i, wf_hole_add[i]);
                printf("Hole%d size: %d KB\n", i, wf_hole_size[i] / 1024);
                printf("----------------------------------------\n");
            }
        }
        printf("\n");
    }
}

/* algo 1-->frist fit   2-->best fit  3-->worst fit
   return 0 means that process has aleady been released
   return 1 means that process can be released */
int release(int pro_num, int algo)
{
    if(algo == 1)
    {
        /* process has been released already */
        if (ff_pro_add[pro_num] == 0)
        {
            return 0;
        }
        int byte_size = ff_pro_size[pro_num] * 1024;
        int i;
        for(i = 0; i < byte_size; i++)
        {
            *(ff_pro_add[pro_num] + i) = 'e';
        }
        ff_pro_size[pro_num] = 0;
        ff_pro_add[pro_num] = NULL;
        return 1;
    }
    else if (algo == 2)
    {
        /* process has been released already */
        if (bf_pro_add[pro_num] == 0)
        {
            return 0;
        }
        int byte_size = bf_pro_size[pro_num] * 1024;
        int i;
        for(i = 0; i < byte_size; i++)
        {
            *(bf_pro_add[pro_num] + i) = 'e';
        }
        bf_pro_size[pro_num] = 0;
        bf_pro_add[pro_num] = NULL;
        return 1;
    }
    else
    {
        /* process has been released already */
        if (wf_pro_add[pro_num] == 0)
        {
```

```
            return 0;
        }
        int byte_size = wf_pro_size[pro_num] * 1024;
        int i;
        for(i = 0; i < byte_size; i++)
        {
            *(wf_pro_add[pro_num] + i) = 'e';
        }
        wf_pro_size[pro_num] = 0;
        wf_pro_add[pro_num] = NULL;
        return 1;
    }
}

int main()
{
    int i;
    int random;
    int stop_indicator;

    /* initialize three block of memories to implement three algoritms */
    char* ff_mem_add = (char*)malloc(MEM_SIZE);
    char* bf_mem_add = (char*)malloc(MEM_SIZE);
    char* wf_mem_add = (char*)malloc(MEM_SIZE);
    for (i = 0; i < 1024 * 1024; i++)
    {
        *(ff_mem_add + i) = 'e'; //e means empty
        *(bf_mem_add + i) = 'e'; //e means empty
        *(wf_mem_add + i) = 'e'; //e means empty
    }

    /* filling process */
    srand(time(NULL));
    while(1)
    {
        random = rand() % 25;  //generate random number [0, 24]
        stop_indicator = fill_up(ff_pro_index, possible_sizes[random], ff_mem_add, 1);
        fill_up(bf_pro_index, possible_sizes[random], bf_mem_add, 2);
        fill_up(wf_pro_index, possible_sizes[random], wf_mem_add, 3);
        ff_pro_index++;
        bf_pro_index++;
        wf_pro_index++;
        if(stop_indicator == 0)
        {
            break;
        }
    }


    /* randomly release 10% */
    int total_process_number = number_of_processes(1);
    int number_needed_to_remove = total_process_number / 10;
    int temp = total_process_number % 10;
    int isremoved;
    srand(time(NULL));
    if(temp >= 5)
    {
        number_needed_to_remove++;
    }
    while(1)
    {
        random = rand() % total_process_number;
        isremoved = release(random, 1);
        release(random, 2);
        release(random, 3);
        if(isremoved == 1)
        {
            number_needed_to_remove--;
        }
        if(number_needed_to_remove == 0)
        {
            break;
```

```c
        }
    }

    /* update hole information after removing 10% */
    update_hole_info(ff_mem_add, 1);
    update_hole_info(bf_mem_add, 2);
    update_hole_info(wf_mem_add, 3);

    /* Display memory information after removing 10% */
    printf("====================Memory Information after removing 10 percent====================\n");
    printf("\n");
    status(ff_mem_add, 1);
    status(bf_mem_add, 2);
    status(wf_mem_add, 3);
    printf("\n");

    /* compaction */
    compact(ff_mem_add, 1);
    compact(bf_mem_add, 2);
    compact(wf_mem_add, 3);

    /* update hole information after compaction */
    update_hole_info(ff_mem_add, 1);
    update_hole_info(bf_mem_add, 2);
    update_hole_info(wf_mem_add, 3);

    /* Display memory information after compaction */
    printf("====================Memory Information after Compaction====================\n");
    printf("\n");
    status(ff_mem_add, 1);
    status(bf_mem_add, 2);
    status(wf_mem_add, 3);
    printf("\n");

    /* refilling the memories */
    int ff_stop_indicator = 1;
    int bf_stop_indicator = 1;
    int wf_stop_indicator = 1;
    srand(time(NULL));
    i = 0;
    while(i <= 10)
    {
        random = rand() % 25;
        allocate(ff_pro_index, possible_sizes[random], ff_mem_add, 1);
        allocate(bf_pro_index, possible_sizes[random], bf_mem_add, 2);
        allocate(wf_pro_index, possible_sizes[random], wf_mem_add, 3);
        i++;
        ff_pro_index++;
        bf_pro_index++;
        wf_pro_index++;
    }

    /* update hole information afte refilling */
    update_hole_info(ff_mem_add, 1);
    update_hole_info(bf_mem_add, 2);
    update_hole_info(wf_mem_add, 3);

    /* Display memory information after refilling */
    printf("====================Memory Information after refilling====================\n");
    printf("\n");
    status(ff_mem_add, 1);
    status(bf_mem_add, 2);
    status(wf_mem_add, 3);

    return 0;
}
```